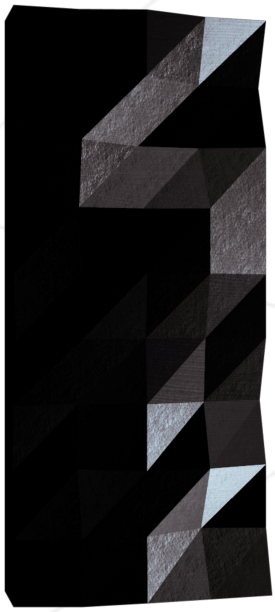

Training Piscine Python for Data Science - 0

Starting

Summary: Today, you will learn the basics of the Python programming language.

Version: 1.3

Contents

Chapter I

General Rules

- You must submit your modules from a computer in the cluster or using a virtual machine:
  - You can choose the operating system for your virtual machine.
  - Your virtual machine must have all the necessary software to complete your project. This software must be installed and properly configured.
- Alternatively, you can use the cluster computers directly if the necessary tools are available.
  - Ensure that you have enough space in your session to install all required dependencies for the modules (use `goinfre` if your campus provides it).
  - Everything must be installed before the evaluations.
- Your functions must not terminate unexpectedly (segmentation fault, bus error, double free, etc.), except in cases of undefined behavior. If such an issue occurs, your project will be considered non-functional and will receive a `0` during the evaluation.
- We encourage you to create test programs for your project, even though these tests **do not need to be submitted and will not be graded**. They will help you easily test your work and your peers' work. These tests will be particularly useful during your defense. During the defense, you are free to use your own tests and/or those of the peer you are evaluating.
- Submit your work to your assigned Git repository. Only the work in the Git repository will be graded. If Deepthought is assigned to grade your work, it will occur after peer evaluations. If an error occurs in any section of your work during Deepthought's grading, the evaluation will be terminated.
- You must use Python version 3.10.
- You may use any built-in function unless explicitly prohibited in the exercise.
- Your library imports must be explicit. For example, you must use `import numpy as np`. Importing libraries using `from pandas import *` is not allowed and will result in a score of `0` for the exercise.

- Global variables are not allowed.
- By Odin, by Thor! Use your brain!!!

Chapter II

Exercise 00

	Exercise 00
Exercice 00: First python script	
Turn-in directory: $ex00/$	
Files to turn in: `Hello.py`	
Allowed functions: `None`	

You need to modify the string of each data object to display the following greetings: "Hello World", "Hello «country of your campus»", "Hello «city of your campus»", "Hello «name of your campus»"

```
ft_list  = ["Hello", "tata!"]
ft_tuple = ("Hello", "toto!")
ft_set   = {"Hello", "tutu!"}
ft_dict  = {"Hello" : "titi!"}

#your code here

print(ft_list)
print(ft_tuple)
print(ft_set)
print(ft_dict)
```

Expected output:

```
$>python Hello.py | cat -e
['Hello', 'World!']$
('Hello', 'France!')$
{'Hello', 'Paris!'}$
{'Hello': '42Paris!'}$
$>
```

Chapter III

Exercise 01

	Exercise 01
Exercice 01: First use of package	
Turn-in directory: $ex01/$	
Files to turn in: `format_ft_time.py`	
Allowed functions: `time, datetime or any other library that allows to receive the date`	

Write a script that formats the dates this way. Of course, your date will not be the same as mine, as in the example, but it must be formatted in the same way.

Expected output:

```
$>python format_ft_time.py | cat -e
Seconds since January 1, 1970: 1,666,355,857.3622 or 1.67e+09 in scientific notation$
Oct 21 2022$
$>
```

Chapter IV

Exercise 02

	Exercise 02
Exercice 02: First function python	
Turn-in directory: $ex02/$	
Files to turn in: `find_ft_type.py`	
Allowed functions: `None`	

Write a function that prints the object types and returns 42.

Here's how it should be prototyped:

```
def all_thing_is_obj(object: any) -> int:
        #your code here
```

Your tester.py:

```
from find_ft_type import all_thing_is_obj

ft_list  = ["Hello", "tata!"]
ft_tuple = ("Hello", "toto!")
ft_set   = {"Hello", "tutu!"}
ft_dict  = {"Hello" : "titi!"}

all_thing_is_obj(ft_list)
all_thing_is_obj(ft_tuple)
all_thing_is_obj(ft_set)
all_thing_is_obj(ft_dict)
all_thing_is_obj("Brian")
all_thing_is_obj("Toto")
print(all_thing_is_obj(10))
```

Expected output:

```
$>python tester.py | cat -e
List : <class 'list'>$
Tuple : <class 'tuple'>$
Set : <class 'set'>$
Dict : <class 'dict'>$
Brian is in the kitchen : <class 'str'>$
Toto is in the kitchen : <class 'str'>$
Type not found$
42$
$>
```

Running your function alone does nothing.

Expected output:

```
$>python find_ft_type.py | cat -e
$>
```

Chapter V

Exercise 03

	Exercise 03
Exercice 03: NULL not found	
Turn-in directory: $ex03/$	
Files to turn in: `NULL_not_found.py`	
Allowed functions: `None`	

Write a function that prints the object type of all types of "Null".
Return 0 if it goes well and 1 in case of error.
Your function needs to print all types of "Null".

Here's how it should be prototyped:

```
def NULL_not_found(object: any) -> int:
 #your code here

```

Your tester.py:

```
from NULL_not_found import NULL_not_found

Nothing = None
Garlic  = float("NaN")
Zero    = 0
Empty   = ""
Fake    = False

NULL_not_found(Nothing)
NULL_not_found(Garlic)
NULL_not_found(Zero)
NULL_not_found(Empty)
NULL_not_found(Fake)
print(NULL_not_found("Brian"))
```

Expected output:

```
$>python tester.py | cat -e
Nothing: None <class 'NoneType'>$
Cheese: nan <class 'float'>$
Zero: 0 <class 'int'>$
Empty: <class 'str'>$
Fake: False <class 'bool'>$
Type not Found$
1$
$>
```

Running your function alone does nothing.

Expected output:

```
$>python NULL_not_found.py | cat -e
$>
```

Chapter VI

Exercise 04

	Exercise 04
Exercice 04: The Even and the Odd	
Turn-in directory: $ex04/$	
Files to turn in: `whatis.py`	
Allowed functions: `sys or any other library that allows to receive the args`	

Create a script that takes a number as an argument, checks whether it is odd or even, and prints the result.

If more than one argument is provided or if the argument is not an integer, print an **AssertionError**.

Expected output:

```
$> python whatis.py 14
I'm Even.
$>
$> python whatis.py -5
I'm Odd.
$>
$> python whatis.py
$>
$> python whatis.py 0
I'm Even.
$>
$> python whatis.py Hi!
AssertionError: argument is not an integer
$>
$> python whatis.py 13 5
AssertionError: more than one argument is provided
$>
```

Chapter VII

From now on you must follow these additional rules

- No code in the global scope. Use functions!
- Each program must have its main and not be a simple script:

```
def main():
  # your tests and your error handling

if __name__ == "__main__":
  main()
```

- Any exception not caught will invalidate the exercises, even in the event of an error that you were asked to test.
- All your functions must have documentation (___doc___)
- Your code must follow the norm
  - pip install flake8
  - alias norminette=flake8

Chapter VIII

Exercise 05

	Exercise 05
Exercice 05: First standalone program python	
Turn-in directory: $ex05/$	
Files to turn in: `building.py`	
Allowed functions: `sys or any other library that allows to receive the args`	

This time you have to make a real autonomous program, with a main, which takes a single string argument and displays the sums of its upper-case characters, lower-case characters, punctuation characters, digits, and spaces.

- If none or nothing is provided, the user is prompted to provide a string.
- If more than one argument is provided to the program, print an **AssertionError**.

Expected outputs:

```
$>python building.py "Python 3.0, released in 2008, was a major revision that is not completely backward
    compatible with earlier versions. Python 2 was discontinued with version 2.7.18 in 2020."
The text contains 171 characters:
2 upper letters
121 lower letters
7 punctuation marks
26 spaces
15 digits
$>
```

Expected outputs: (the carriage return counts as a space, if you don't want to return one use ctrl + D)

```
$>python building.py
What is the text to count?
Hello World!
The text contains 13 characters:
2 upper letters
8 lower letters
1 punctuation marks
2 spaces
0 digits
$>
```

By Odin, by Thor ! Use your brain !!! Don't reinvent the wheel, use the language features.

Chapter IX

Exercise 06

	Exercise 06
Exercice 06:	
Turn-in directory: $ex06/$	
Files to turn in: `ft_filter.py, filterstring.py`	
Allowed functions: `sys or any other library that allows to receive the args`	

Part 1: Recode filter function

Recode your own ft_filter, it should behave like the original built-in function (it should return the same thing as "print(filter.___doc___)"), you should use **list comprehensions** to recode your ft_filter.

`Of course using the original filter built-in is forbidden`

`You can validate the module from here, but we encourage you to continue as there are things you will need to know for the following projects`

Part 2: The program

Create a program that accepts two arguments: a string (S) and an integer (N). The program should output a list of words from **S** that have a length greater than **N**.

- Words are separated from each other by space characters.
- Strings do not contain any special characters (punctuation or invisible).
- The program must contain at least one **list comprehension** expression and one **lambda**.
- If the number of arguments is different from 2, or if the type of any argument is wrong, the program prints an **AssertionError**.

Expected outputs:

```
$> python filterstring.py 'Hello the World' 4
['Hello', 'World']
$>
```

```
$> python filterstring.py 'Hello the World' 99
[]
$>
```

```
$> python filterstring.py 3 'Hello the World'
AssertionError: the arguments are bad
$>
```

```
$> python filterstring.py
AssertionError: the arguments are bad
$>
```

Chapter X

Exercise 07

	Exercise 07
Exercice 07: Dictionaries SoS	
Turn-in directory: $ex07/$	
Files to turn in: `sos.py`	
Allowed functions: `sys or any other library that allows to receive the args`	

Make a program that takes a string as an argument and encodes it into Morse Code.

- The program supports space and alphanumeric characters.
- An alphanumeric character is represented by dots `.` and dashes `-`.
- Complete Morse characters are separated by a single space.
- A space character is represented by a slash `/`.

You must use a **dictionary** to store your morse code.

```
NESTED_MORSE = {   " ": "/ ",
                   "A": ".- ",
                   ...
```

If the number of arguments is different from 1, or if the type of any argument is wrong, the program prints an **AssertionError**.

```
$> python sos.py "sos" | cat -e
... --- ...$
$> python sos.py 'h$llo'
AssertionError: the arguments are bad
$>
```

Chapter XI

Exercise 08

	Exercise 08
Exercice 08: Loading ...	
Turn-in directory: $ex08/$	
Files to turn in: `Loading.py`	
Allowed functions: `os`	

So let's create a function called `ft_tqdm`.
The function must copy the function tqdm with the `yield` operator.

Here's how it should be prototyped:

```
def ft_tqdm(lst: range) -> None:
 #your code here
```

Your tester.py: (you compare your version with the original)

```
from time import sleep
from tqdm import tqdm
from Loading import ft_tqdm

for elem in ft_tqdm(range(333)):
        sleep(0.005)
print()
for elem in tqdm(range(333)):
        sleep(0.005)
print()
```

Expected output: (you must have a function as close as possible to the original version)

```
   $> python tester.py
100%|[============================================================>]| 333/333
100%|                                                               | 333/333 [00:01<00:00, 191.61it/s]
```

`You can use get_terminal_size to adapt to the size of your terminal.`

Chapter XII

Exercise 09

	Exercise 09
Exercice 09: My first package creation	
Turn-in directory: $ex09/$	
Files to turn in: `*.py, *.txt, *.toml, README.md, LICENSE`	
Allowed functions: `PyPI or any library for creation package`	

Create your first package in python the way you want, it will appear in the list of installed packages when you type the command "pip list" and display its characteristics when you type "pip show -v ft_package"

```
$>pip show -v ft_package
Name: ft_package
Version: 0.0.1
Summary: A sample test package
Home-page: https://github.com/eagle/ft_package
Author: eagle
Author-email: eagle@42.fr
License: MIT
Location: /home/eagle/...
Requires:
Required-by:
Metadata-Version: 2.1
Installer: pip
Classifiers:
Entry-points:
$>
```

The package will be installed via pip using one of the following commands (both should work):

- pip install ./dist/ft_package-0.0.1.tar.gz
- pip install ./dist/ft_package-0.0.1-py3-none-any.whl

Your package must be able to be called from a script like this one:

```
from ft_package import count_in_list

print(count_in_list(["toto", "tata", "toto"], "toto")) # output: 2
print(count_in_list(["toto", "tata", "toto"], "tutu")) # output: 0
```

Chapter XIII

Submission and peer-evaluation

Turn in your assignment in your `Git` repository as usual. Only the work inside your repository will be evaluated during the defense. Don't hesitate to double check the names of your folders and files to ensure they are correct.

`The evaluation process will happen on the computer of the evaluated group.`